

1/5

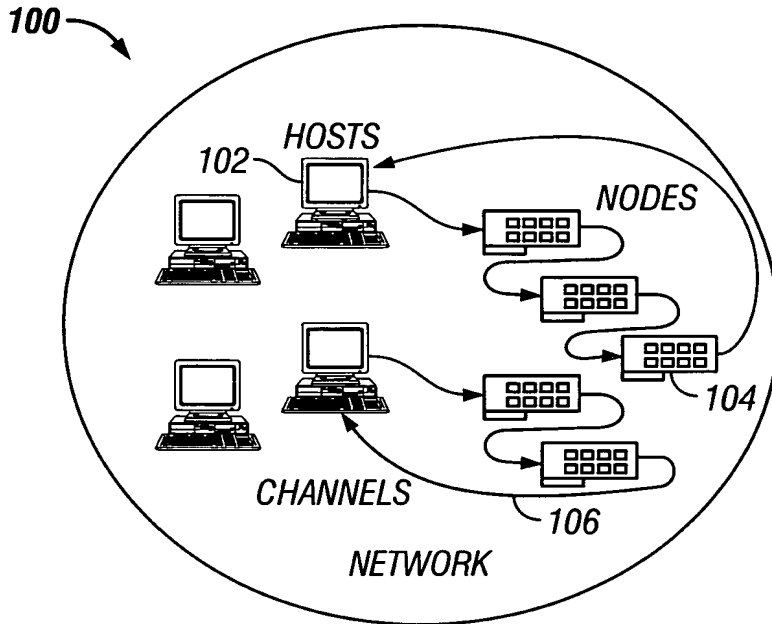


FIG. 1

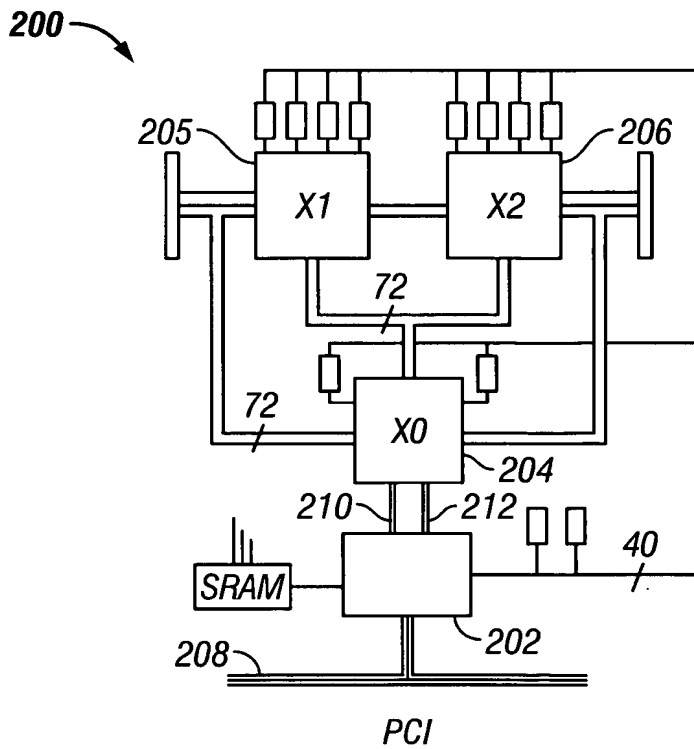


FIG. 2

2/5

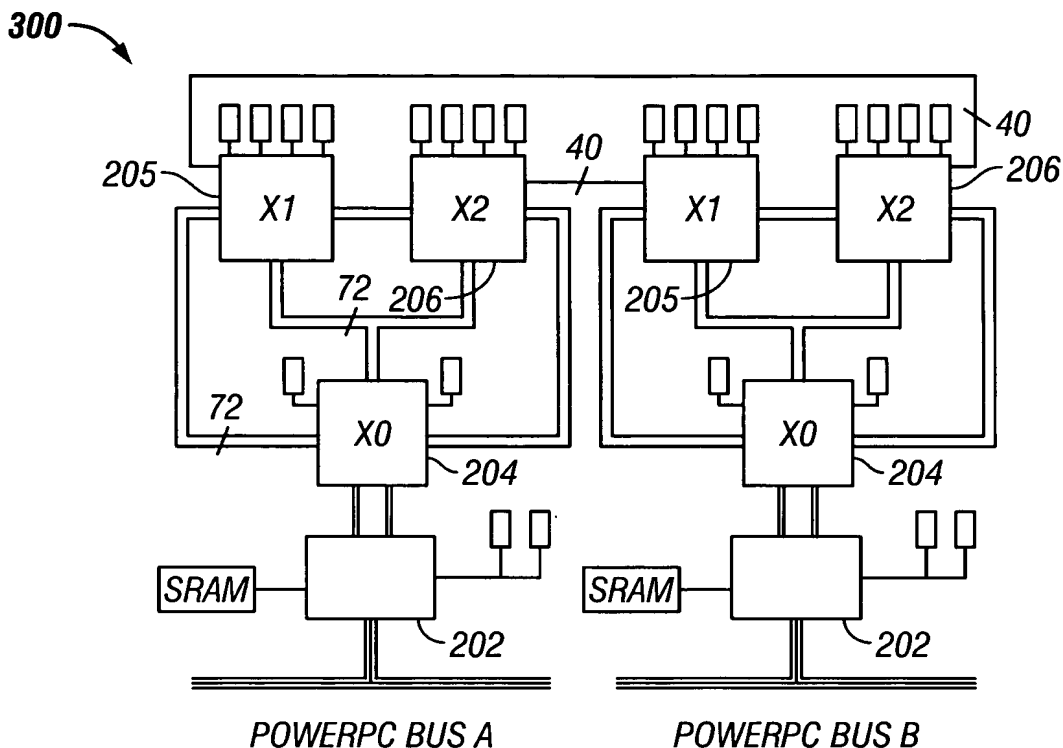


FIG. 3

400

```

ACS_NODE nodes[4]; /* user structure to describe nodes */
ACS_CHANNEL channels[4]; /* user structure for channels */
ACS_STATUS status; /* status of ACS API commands */
ACS_SYSTEM ring;

for (int i=0;i<4;i++) { /* build a ring of 4 WildForce boards */
    nodes[i].model = "WF4";
    channels[i].src_node = i;
    channels[i].src_port = 0;
    channels[i].dest_node=(i+1)%4;
    channels[i].dest_port=1;
}
ACS_Initialize(argc, argv, &status); /* must be first API call */
ACS_System_Create(&ring, nodes, 4, channels, 4);
/* user program that accesses "ring" object */
ACS_System_Destroy(ring);
ACS_Finalize(); /* must be last API call */
    
```

FIG. 4

3/5

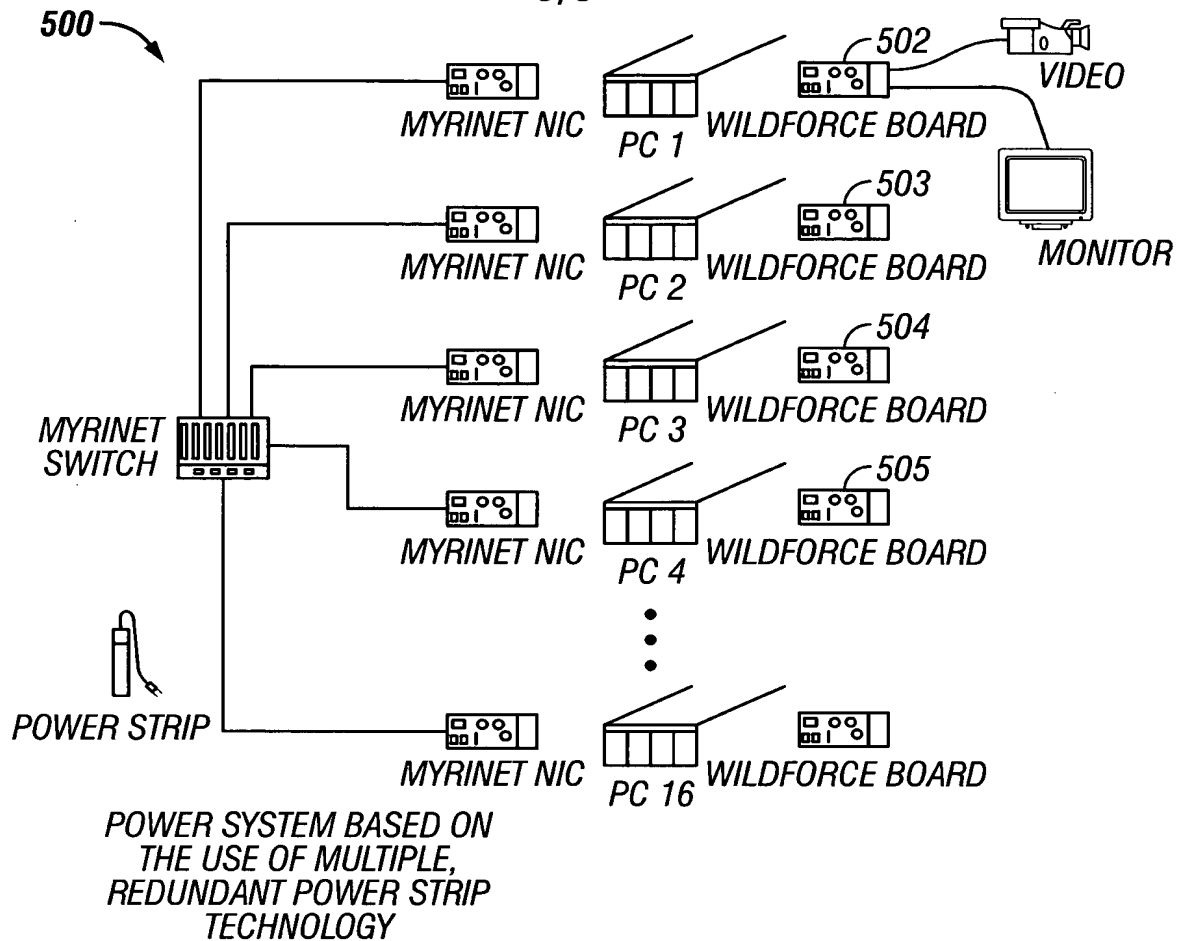


FIG. 5

600

```

for (int i=0;i<4;i++) {
    /* send bitstream for each ACS board */
    ACS_Configure(config[i],i,ring,&status);
    ACS_Clock_Set(clock,i,ring,&status);/* set clock speed */
    ACS_Run(i,ring,&status);/* start clock */
    ACS_Reset(i,ring,&status);/* send reset signal */
}
for (int i=0;i<4;i++) {
    /* write initial data to each board's memory */
    ACS_Write(databuf[i],datalen[i],i,brd_addr[i],ring,&status);
    /* then send an interrupt (or inta) signal #1 to the baord */
    ACS_Interrupt(i,1,ring,&status);
}
    
```

FIG. 6

4/5

700

```
/* use the ring to process the required number of images */  
for (int i=0;i<NUM_IMAGES;i++) {  
    /* send image onto channel associated with port 0 */  
    ACS_Enqueue(image[i],IMAGESIZE,0,ring,&status);  
    /* get resulting image from channel associated with port 1 */  
    ACS_Dequeue(result_image[i],RESULT_SIZE,1,ring,&status);  
}
```

FIG. 7

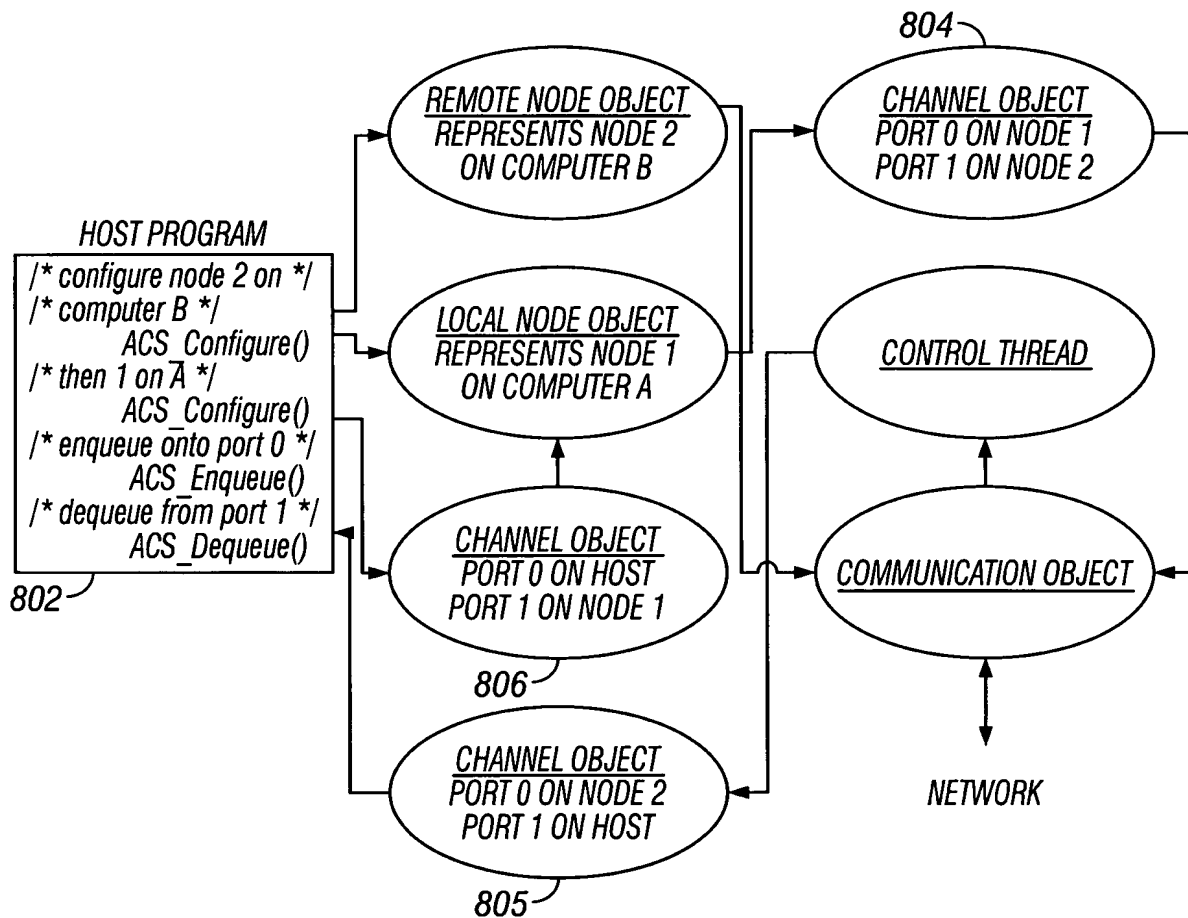


FIG. 8A

5/5

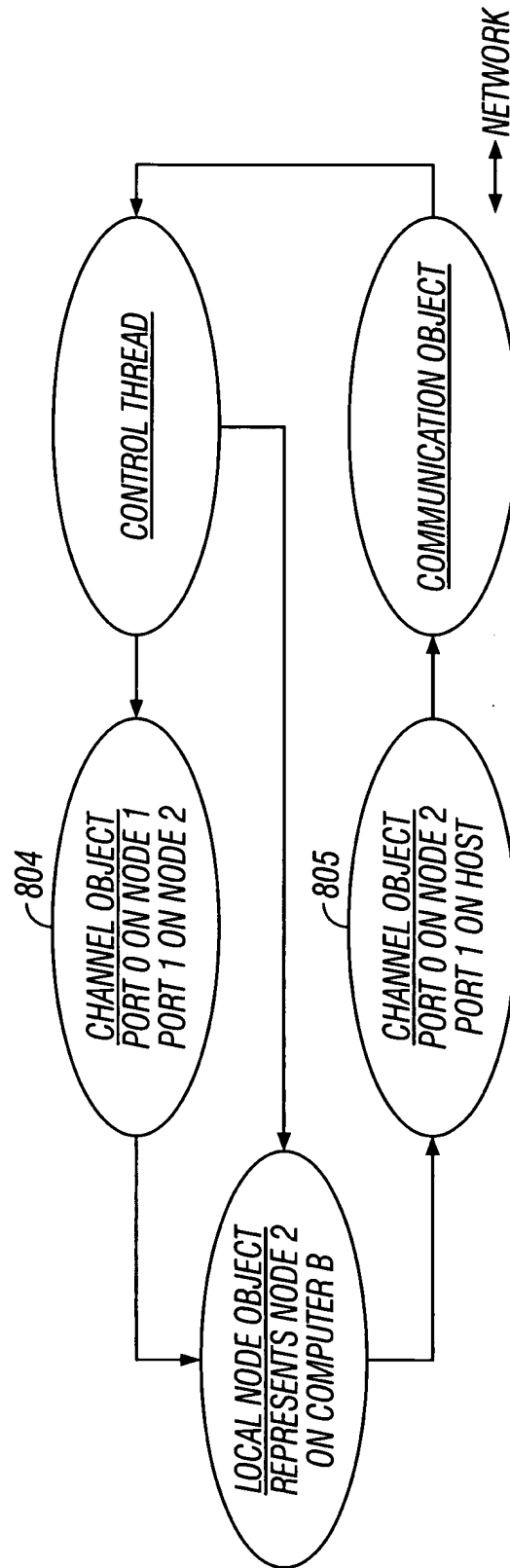


FIG. 8B